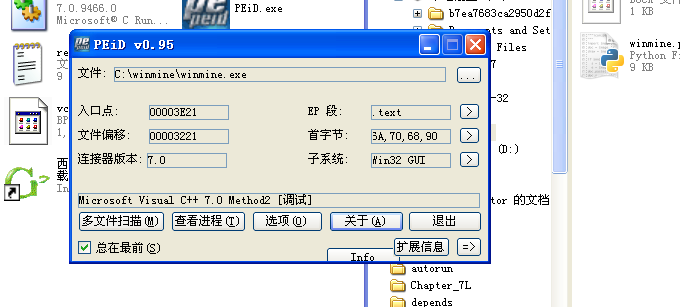
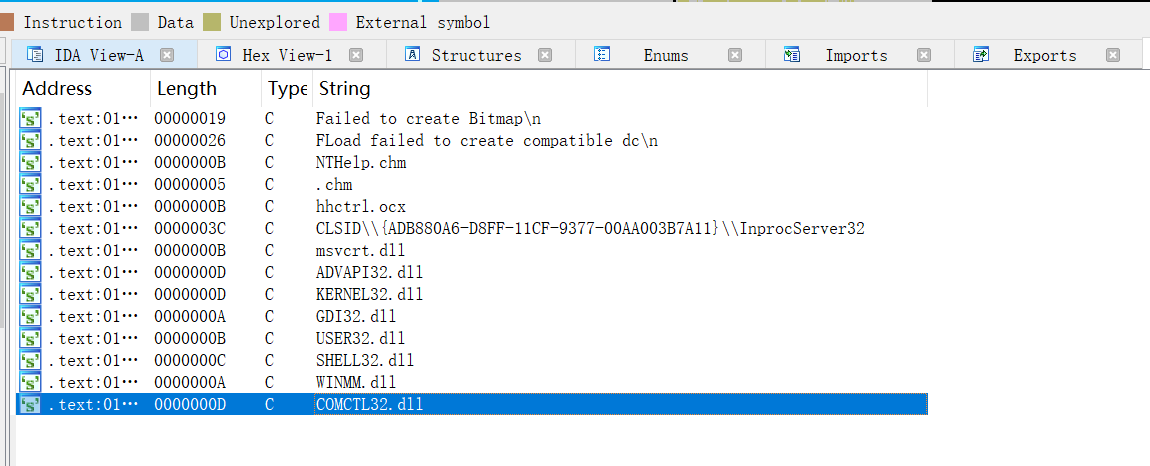
# Windows xp下winmine基本分析

1. 对于软件进行查壳->防止软件加壳



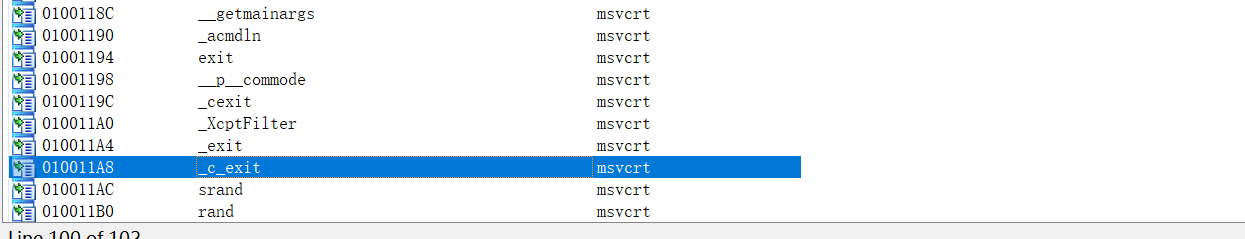
可以看到程序没有进行加壳，并且是debug的模式，继续向下分析，将程序放在ida中以及od中分别进行分析

1. 放在ida中查看程序中对应的字符串之类的可以用来分析程序的主函数之类的相关信息

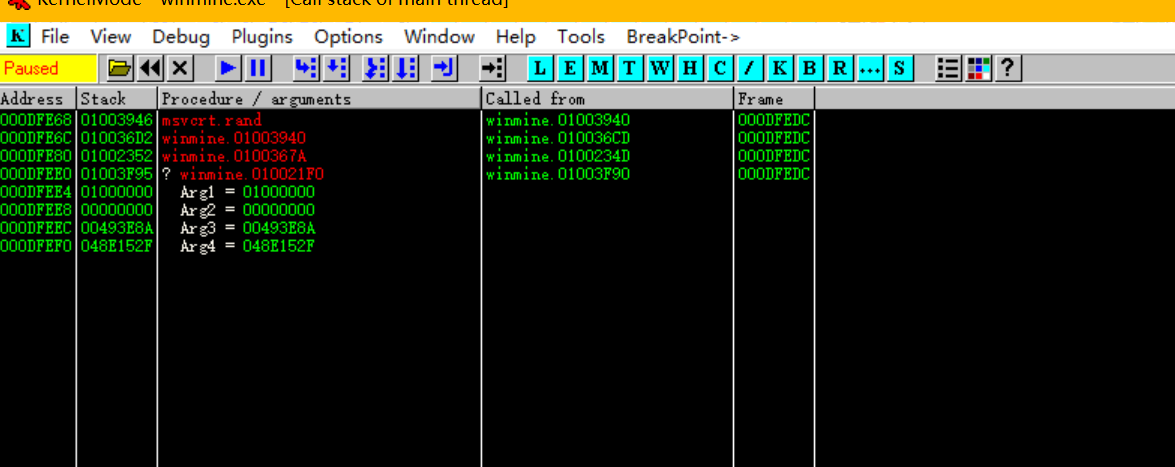


大概分析了一下，看到其中对应的字符串和函数的真正执行流程有点不怎么相关，接下来继续进行分析，能不能通过点击按键之类的找一下相关的main函数和字符串

1. Python实现自动扫雷的算法
   1. 拿到雷区对应的数据，模拟鼠标的动作，点击雷区上非雷的格子
   2. 雷区对应的内存中的数据在哪里，直接通过获取雷区的图片然后分析图片拿到的？？？
      1. Winxp中是不是不开启ASLR, 所以程序中加载的都在固定的位置，所以能拿到
      2. 扫雷函数对应的主函数在什么位置，怎么拿到扫雷对应的主函数-》下断点开始执行，或者能不能ida attach到已知的进程上：
   3. 根本不知道程序的入口地址在哪里，所以参照教程<https://blackbinary.net/?p=503进行向下的逐步分析> -》 使用ollydbg加载并且拿到对应的函数
2. 首先是使用ctrl+g对于rand函数下断点（因为考虑的是函数进行初始化的时候采用随机雷区，所以可能使用rand函数



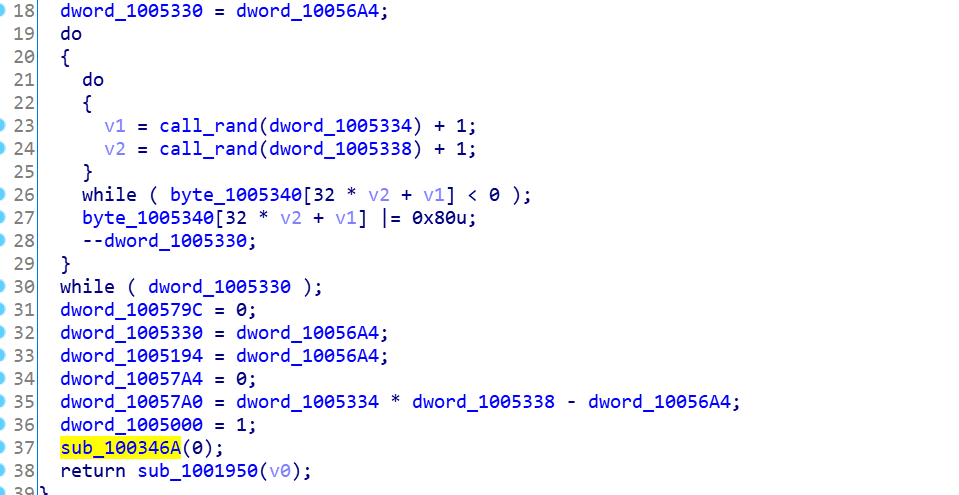
可以发现ida中能看到rand以及srand，为什么不搜索出来srand并且下断点分析callstack中的调用情况



Od中的函数调用情况是020021f0 -> 可以直接在ida中查看到相同的地址，继续对于地址向下分析

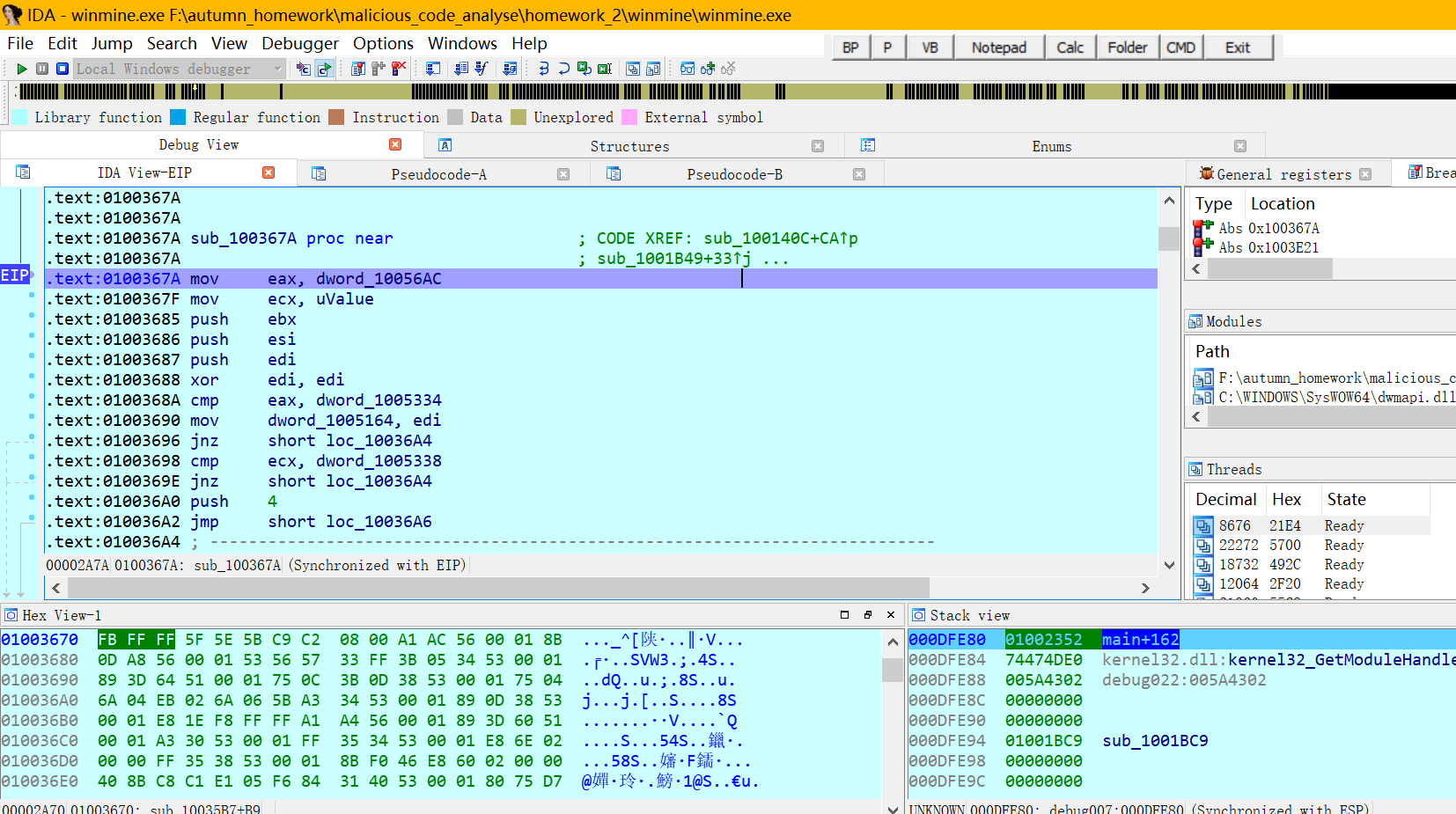
差不多就是直接从callstack中往前传递，可以发现在winmine.01003940的位置通过调用了rand()函数 –》 能不能认为这个就是对应的内存的位置，然后查看上下的汇编实现了解到扫雷是怎么进行分布的. 所以应该在ida中跟踪什么函数才能认为是进行了扫雷的分布的

* + - * 0100367a 中调用了01003940函数 ，01003940函数的作用是调用rand函数实现了对于布局的随机化 –》 反汇编看函数的基本流程
      * 直接找雷区对应的数组的位置，通过反汇编0100367a实现看这个函数中的大概的逻辑，因为猜测的实现逻辑是先用rand拿到一个随机数，然后通过使用随机数实现对于程序中的雷区的位置进行赋值，



感觉程序中比较关键的数据结构在这个位置，首先是v1,v2通过rand函数进行初始化，对于其中的数据追溯一下发现，byte\_1005340是一个char型的数组，直接看函数的逻辑应该是分析这个数组byte\_1005340[32\*v2+v1]<0 的时候就进行赋值，那应该是byte\_1005340中本身就有数据才能这样判断，然后设置了一个标志位0x80u。外层也套了一个对应的do while的数据结构，要是能识别出来dowrd\_1005330是和雷的数目相关的话，其实就能确定出来byte\_1005330中存放了雷，就能定位到存雷的地址空间位置了。

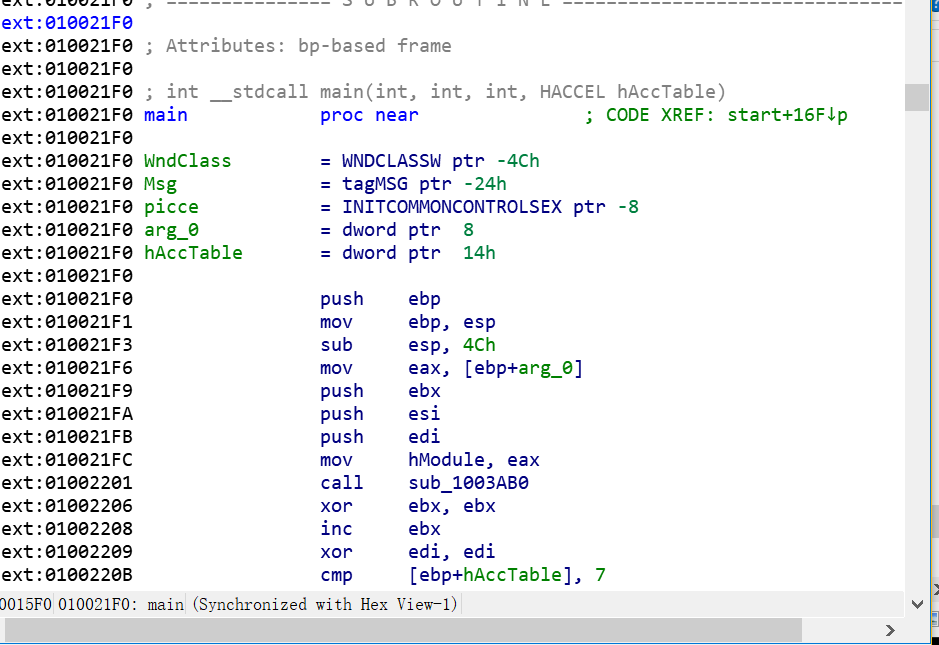
因为这个部分已经能定位到大概的雷产生的情况了，所以通过ida进行动态调试分析其中的执行的流程，



通过在反汇编的代码进行调试的时候发现dowrd\_1005330 -> 10 这个就是初级条件下的雷的数目，所以能猜测出来byte\_1005340中应该存放了雷的情况，继续向下调试分析。感觉byte\_1005340是不是就是一个内存的开始的位置，其中存放雷的索引是用v1,v2两个随机数确定的，也不知道byte\_1005340最后对应的存雷的区间到底有多大，感觉还是有点难。

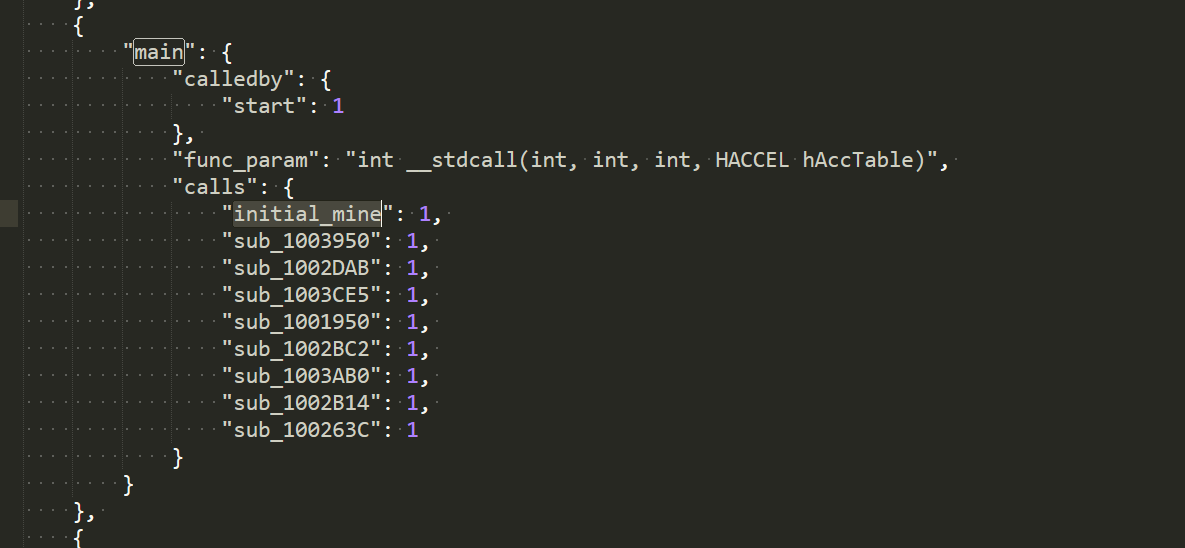
到目前位置可以总结一下，对应的雷区的内存的开始的位置是0x1005340. 然后通过v1= rand(9)+1, v2 =rand(9)+1 map[32\*v1+v2] = 0x80 就认为这个对应的内存中有雷 –》感觉分析出来了这个数组，只要能拿到这个数组对应的区域的值jiu能拿到游戏中的雷的布局。

1. 继续向下分析，当前分析的这个函数是从ollydbg中看到的，这个函数从callback中可以分析出来，这个函数是程序执行的过程中第一个被调用的函数，所以认为这个函数是主函数，从主函数向下进行分析函数大概的逻辑，分析过程中需要找到1. 时钟的位置，2，点到雷结束游戏的函数



本身时钟所在的位置应该是在菜单栏，而且是第一次点击之后才能初始化时钟，所以想了两个思路，第一个是从主函数中向下看函数被调用的情况，然后分析哪个函数中可能有时钟，第二个是想定位到time这种函数，直接用动态的方法拿到时钟的地址

1. 还是继续回去分析主函数，先拿到了一下主函数的调用函数的情况



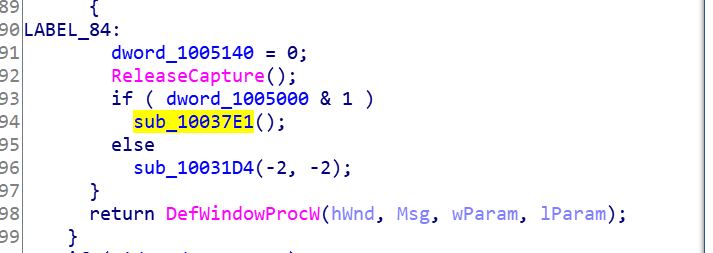
所以相对于主函数中调用的函数进行分析看看能不能拿到clock相关的函数：

首先对于主函数中调用的系统函数进行分析：-》定位一下函数中的大概的逻辑做的事情

1. InitCommonControlsEx: 确保加载公共控件DLL（Comctl32.dll），并从DLL注册特定的公共控件类。应用程序必须在创建公共控件之前调用此函数。
2. LoadIconW : 加载图标相关的资源
3. **lpfnWndProc：**是一个指向应用程序定义函数的指针，该函数称为*窗口过程*或“窗口*过程* ”。窗口过程定义了窗口的大部分行为。我们稍后会详细检查窗口过程。现在，只需将其视为前向参考（备注：这个是定义了一个窗口的相关行为，所以之前参考了一个教程，上面说这个可能是和扫雷的相关行为有关，要对于这个后面的函数详细分析）
4. RegisterClassW：主要是用来后面规定窗口中对应的相关行为
5. LoadCursorW：从与应用程序实例关联的可执行文件（.EXE）加载指定的游标资源。
6. GetStockObject：这个函数句柄用来检索stock pens, brushes, fonts, or palettes.
7. LoadMenuW：从与应用程序实例关联的可执行（.exe）文件加载指定的菜单资源。
8. LoadAcceleratorsW：加载指定的加速器表。
9. CreateWindowExW：函数的作用应该是创建一个窗口，创建出来的窗口就是扫雷的窗口
10. DefwindowsProcW：
11. 分析lpfnwndproc：这个函数定义了一个窗口的事件回调函数，通过对于回调函数分析可以发现其中规定的相关的行为，

对应的函数是sub\_1001bc9: 对于这个函数进行分析，其中的msg定义了鼠标的击键相关行为，查询之后发现0x202U表示的鼠标的信号是：WM\_LBUTTONUP 所以对应的跳转的标签是键盘的击键消息。

所以接下来分析label\_84对应的键盘上的标志位：



首先是一个函数消息的调用-》想直接下断点进行执行，查看其中对应的函数的逻辑，

1. 系统调用函数：releasecapture：这个函数标识的意思是：从当前线程中的窗口释放鼠标捕获并恢复正常的鼠标输入处理。无论光标位置如何，捕获鼠标的窗口都会接收所有鼠标输入，除非在光标热点位于另一个线程的窗口中时单击鼠标按钮。对这部分代码进行动态调试分析：
2. 发现要是有按键，就会进入到sub\_10037e1的函数中。开始对于这个子函数调用进行分析：
3. 函数中首先能看到一个set\_timer函数，这个函数的作用时设置计时器，所以可以猜一下是不是执行到这个函数的时候开始进行计时。（调试的时候想了一下，是不是数据段中的变量都是全局变量，然后使用全局变量进行标识，比方说dword\_100579c是不是一个全局变量用来表示第几个格子被翻开什么的，）
4. 继续向下进行分析，发现执行了get\_dc\_and\_release\_dc子函数的时候，会出现计时器增加了1位，但是增加了之后不再继续增加，不知道是不是能从这个地方找到计时器对应的内存地址 –》 在sub\_1002825中找内存感觉有点麻烦，继续向下分析，看一下set\_timer中能不能进去调试找到计时器的内存

SetTimer函数相当于设置了一个定时器，感觉和秒数的增加不是很相关，所以需要重新启动线程到sub\_1002825对应的内存中进行查找，查一下是不是能找到内存变化点，发现settimer函数还有上面的函数只有第一次执行的时候才能进入，下面的函数的分析过程相当的复杂

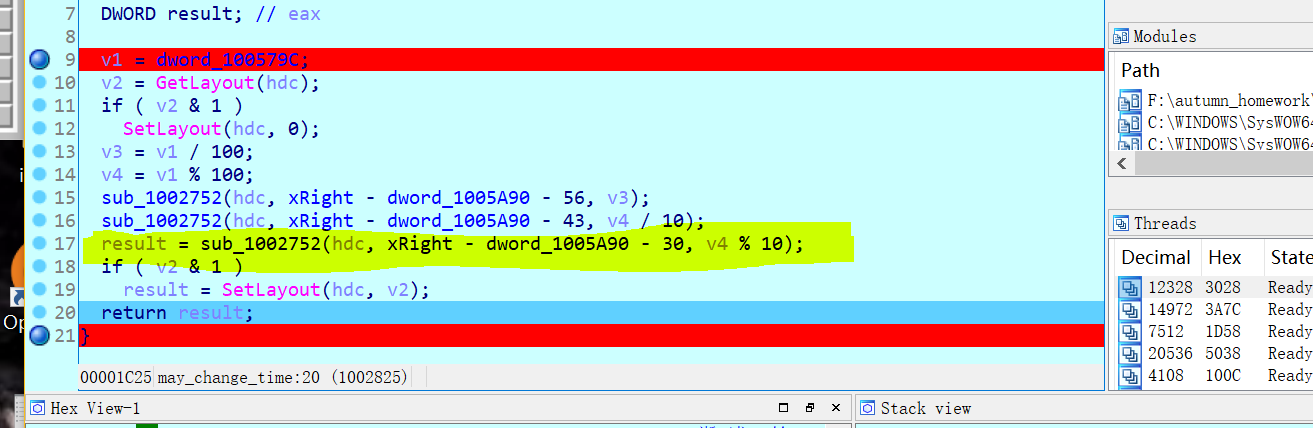
再次返回去执行get\_dc\_and\_release\_dc的函数时候分析，发现中间通过执行sub\_1002752的时候调用了一个库函数，而且return有返回值的时候感觉计时器发生了一次变化。

**SetDIBitsToDevice：这个库函数的作用是**功能设置与使用从DIB，JPEG，或PNG图像彩色数据的目的地装置上下文相关联的设备上的指定的矩形的像素。设置图像像素是改变图像吗

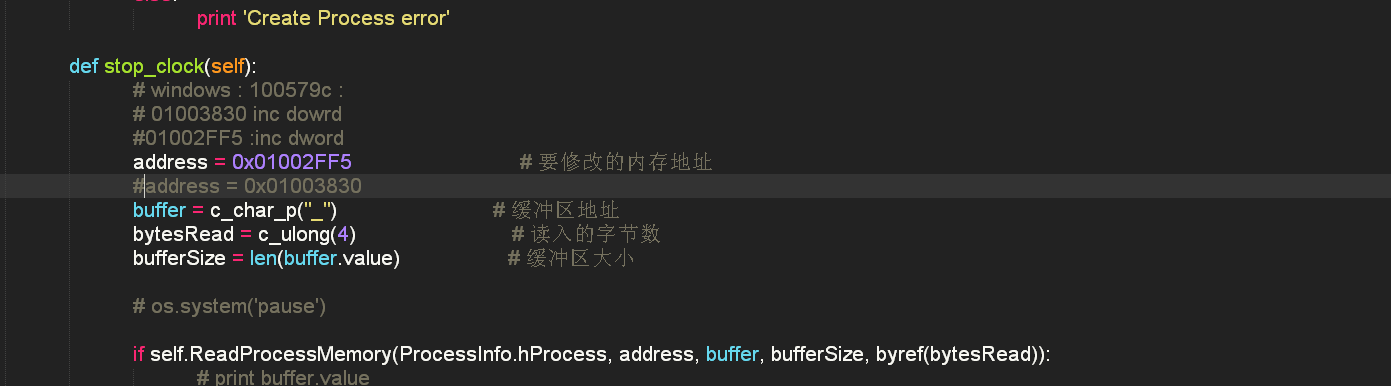
**Setlayout:** **所述的setLayout函数更改设备上下文（DC）的布局。**

是不是这个函数中间发生了变化，拿到了计数的位置对应的内存？？

这个对于计数的分析陷入僵局-》查了一下教程来分析实现的方案

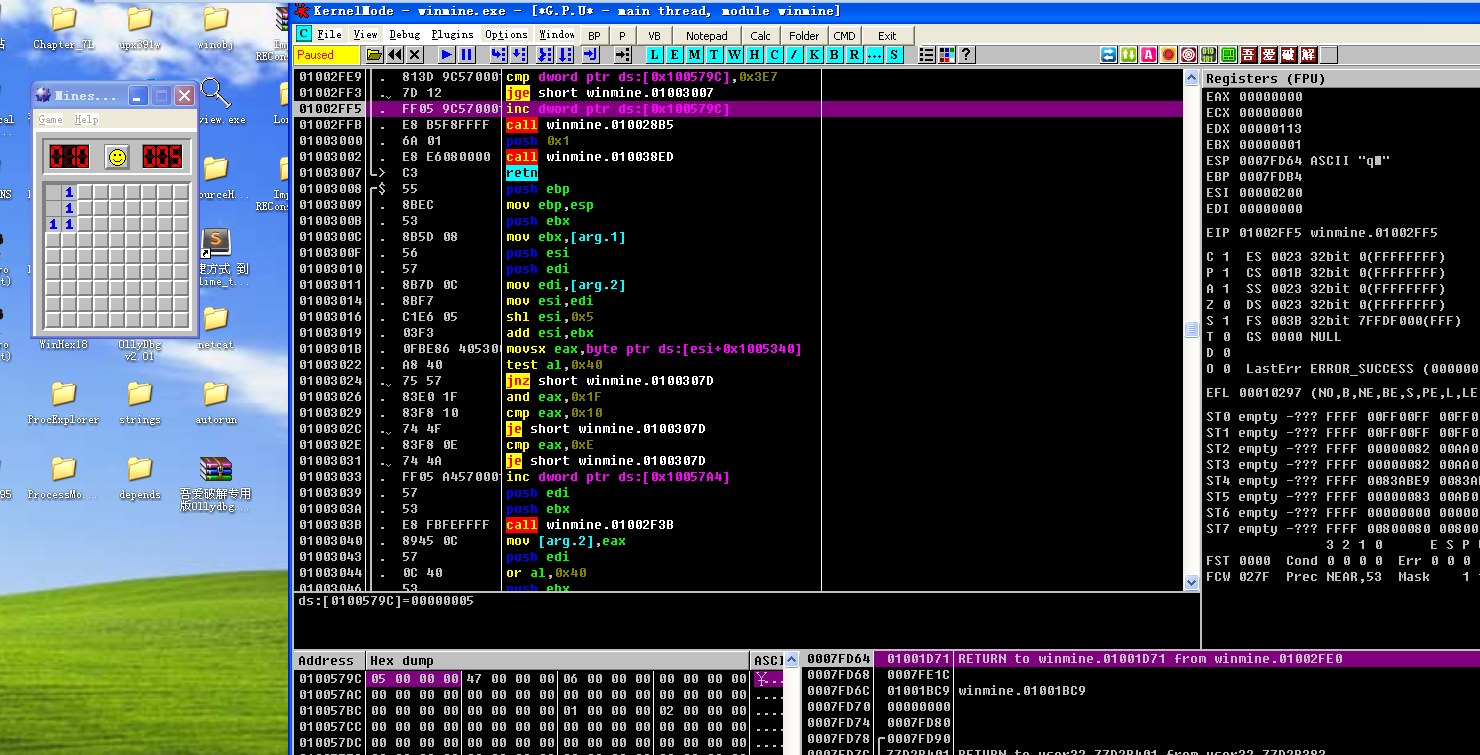


后面调试的时候能发现对应的扫雷程序中的时间应该是在dowrd\_100579c中间，怎么才能打开一个进程之后让这个值保持不变，直接程序中可以通过nop掉指令实现扫雷不进行计数。



所以通过xref这个对应的地址发现；inc 这个地址的地方有两个，分别如下所示，两个地址进行测试，发现01003830对应的地址nop之后还是会有计数器的改变，非常不理解了，明显在发生按键的时候应该是有

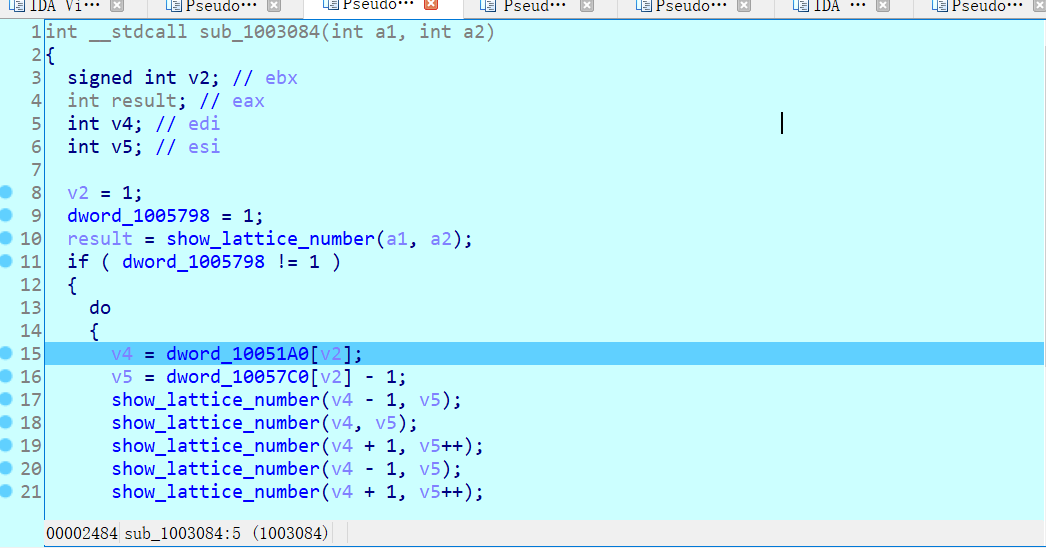
对于上面的这个问题的疑问应该通过od调试实现，但是发现好像断到了od的对应的位置之后，好像不能像教程那样会停止还能下一次执行，还是说我设置的内存的断点不正确，而且ida调试的话应该也能调试出来的吧，为啥他们就能知道对应的地址是0x1002ff5这个位置设置成nop, 突然想到另外一种可能，是不是这个部分只能执行一次，所以程序执行之后就不会再返回回来执行了，还有就是od的执行机制比较奇怪吗，下了断点就会pause，然后需要重新点击run，不是点击continue实现继续分析



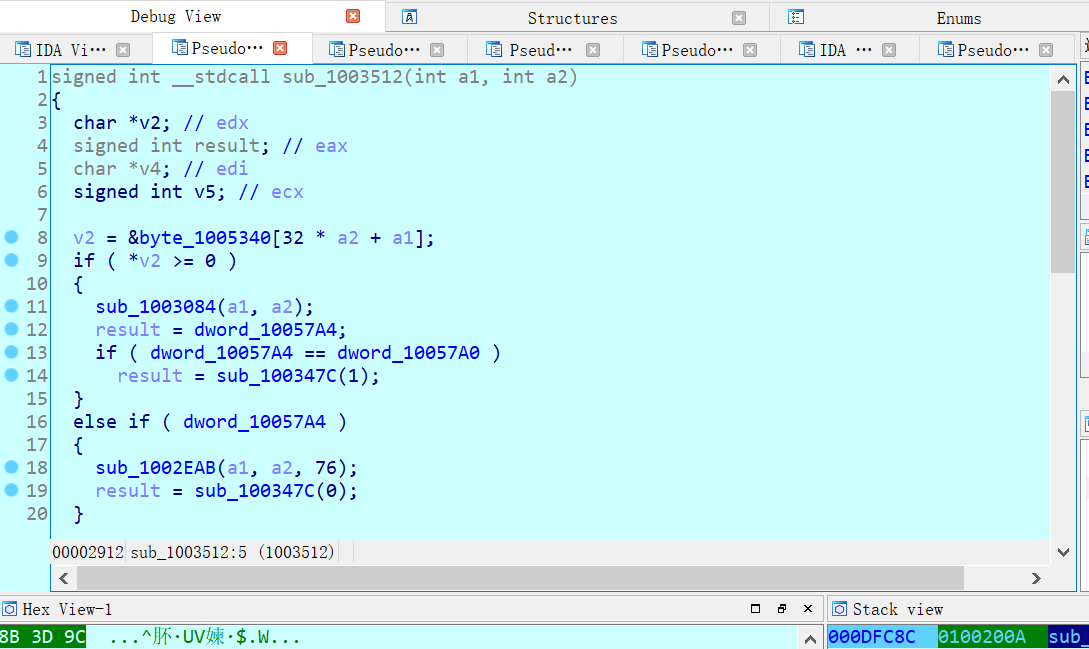
继续分析：通过在ida中动态调试发现，1002ff5的这个地址是每次计数器发生改变的时候都会访问的地址，之前的3xxx的地址好像只有第一次访问的时候才改变，所以确实应该通过nop掉这个位置实现 🡨-------对于时间的分析结束

补充分析：

1. 从当前的函数的断点向下进行分析：-》按键按下之后-》时钟发生第一次改变，然后就转到了show\_lattrace\_number：判断当前的格子对应的数据是什么

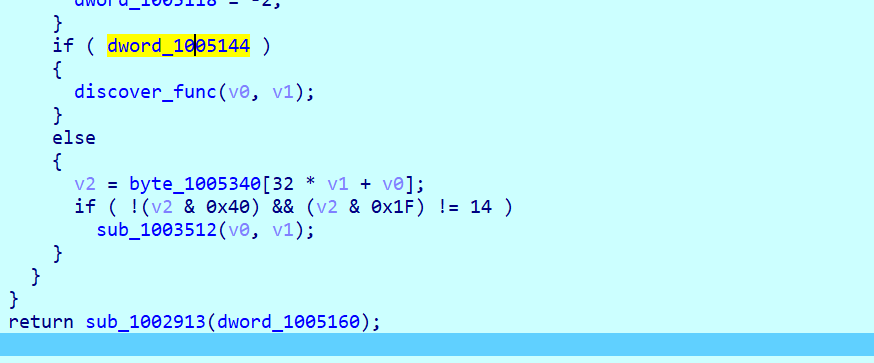


感觉这个函数的作用是从当前位置向左右上下游走，判断对应的格子中的结果



对于3512这个进行分析，还是感觉能明显找到byte\_1005340中间存储了雷的状态，继续往下分析一步，这个好象的作用就是第一次点击之后，进到这个函数中间，

1. 空白格子：点到空白格子就继续往周围找直到点到正常的数字的格子停止
2. 踩雷格子：
3. 数字的格子：点到数字的格子就停止向下点，停在了数字的格子上、
4. 是不是应该返回去分析一下onpressbutton函数了，



感觉中间调用的时候标黄的数据来判断是不是第一次踩雷，要是第一次踩到雷的话也会对雷进行更新保证程序继续执行，

3.

1. 继续向下分析：分析扫雷的时候怎么才能显示对应的雷的布局，拿到对应的内存的地址，然后通过分析内存的地址了解整个对应的雷的布局？？
2. 点到雷不会停止，应该是有相关的函数实现整个功能, 所以应该还是回去找对应的函数的实现